Sequential Aggregate MACs from Any MACs: Aggregation and Detecting Functionality

Shingo Sato[1], Shoichi Hirose[2], and Junji Shikata[1]*
[1]Yokohama National University, Yokohama, Japan
sato-shingo-cz@ynu.jp, shikata-junji-rb@ynu.ac.jp
[2]University of Fukui, Fukui, Japan
hrs_shch@u-fukui.ac.jp

Abstract

The aggregate message authentication code (aggregate MAC) is a cryptographic primitive which can compress MAC tags on multiple messages into a short aggregate MAC tag. Furthermore, the sequential aggregate MAC can check not only the validity of multiple messages but also the (sequential) order of messages. In this paper, we introduce a new model of sequential aggregate MACs (SAMACs) where an aggregation algorithm generates a sequential aggregate tag depending only on any multiple and independent MAC tags with no secret-key, and we formally define security in this model. We also propose a generic construction of sequential aggregate MACs starting from any MACs without changing the structure of the MACs. This property is useful to make the existing networks more efficient by combining the aggregation algorithm with various MAC schemes already existing in the networks. Furthermore, by extending the results of SAMAC, we also introduce a sequential aggregate MAC which has detecting functionality (SAMD). The SAMD enables us to specify an invalid message or an invalid order of a certain message. We formalize the security of SAMD and provide a generic construction of SAMD in the random oracle model from any MACs and non-adaptive group testing protocols with formal security proofs.

Keywords: Message authentication, MAC, Aggregate MAC, Sequential aggregate MAC

1 Introduction

1.1 Background

The message authentication code (MAC) is one of the most fundamental cryptographic primitives. Furthermore, Katz and Lindell [11] proposed the aggregate MAC that can compress multiple MAC tags on multiple messages generated by different signers into a single aggregate tag. The advantage of the aggregate MAC lies in that the size of an aggregate tag is much smaller than total sizes of MAC tags, and hence it will be useful in applications in mobile networks or IoT (Internet of Things) networks where many devices sending messages are connected. The model and security of aggregate MACs were introduced by Katz and Lindell [11], and they proposed the simple construction satisfying the security by using exclusive-or of MAC tags.

Furthermore, there is another line of research about compressing MAC tags, called the sequential aggregate MACs. In sequential aggregate MACs, we can check not only the validity of multiple messages (like the aggregate MACs) but also the (sequential) order of messages. This property is required in applications including networks of resource-constrained devices such as IoT networks and mobile ad-hoc networks (MANET). Eikemeier et al. [6] formally defined the model and security for sequential

Journal of Internet Services and Information Security (JISIS), volume: 9, number: 1 (February 2019), pp. 2-23
*Corresponding author: Graduate School of Environment and Information Sciences, Institute of Advanced Sciences, Yokohama National University, 79-7 Tokiwadai, Hodogaya-ku, Yokohama 240-8501, Japan, Tel: +81-45-339-4340

aggregate MACs. They also introduced history-freeness which is a property depending only on a local message of each sender and the prior aggregate tag (aggregate-so-far tag), and they proposed history-free sequential aggregate MAC schemes. Ma and Tsudik [13] gave a simple construction by using hash functions for sequential aggregate MACs with forward security, however, they did not give a formal security proof to show that their construction met the security. Hence, Hirose and Kuwakado [8] formally defined the forward security in sequential aggregate MACs, and proposed a construction satisfying the security property with a formal security proof. Tomita et al. [18] gave a model of sequential aggregate authentication codes in the information-theoretic security setting, and they proposed constructions along with their model. The model in [18] focuses the one-time information-theoretic security which is different from those of [6, 13, 8].

Our motivation is to make the existing networks using MACs more efficient than the present state of affairs, however, it is not realistic to replace the currently existing network protocols with other ones entirely in general. Instead, we consider to simply embed a new node for improvement of efficiency (by aggregating MAC-tags sequentially) into the existing network without changing input-formats or structures of the existing MACs in the networks. In this paper, we call such a node an *aggregate node* whose role is to sequentially compress any multiple MAC-tags into a short tag without managing secret keys. The prior work for sequential aggregate MACs [6, 8, 13] does not satisfy our targeted property, namely, the prior work needs a new system setting (e.g., changing composition of MACs or setting an aggregate algorithm with a secret-key) or needs to change input-formats of the underlying MAC schemes (e.g., additional information with the local message would be required as input of MACs).

1.2 Our Contribution

In this paper, we introduce a new model of sequential aggregate MACs (SAMACs) where an aggregation algorithm generates a sequential aggregate tag depending only on multiple and independent MAC tags without any secret-key, and we formally define security in this model. Our model and security are quite different from those of previous works. In addition, we propose two generic constructions of sequential aggregate MACs, called SAMAC1 and SAMAC2, starting from any MAC schemes (e.g., HMAC [1, 2], CMAC [14]), and we formally prove that our constructions meet the security.

To clarify the (dis)advantage of our constructions, we compare ours (i.e., SAMAC1 and SAMAC2) and the existing ones (i.e., MT[13], EFG+[6], and HK[8]) in terms of *universal applicability*, *security*, and *efficiency* in Tables 1-3, where we do not compare TWS[18] with others, since the security of TWS is information-theoretic and quite different from others. In the following, we explain differences among them by using Tables 1-3.

(i) **Universal Applicability.** We consider applicability of embedding an *aggregate node* into the existing MAC protocols without changing the input-formats or network connections of underlying MACs. We also consider whether an aggregate algorithm can be executed without secret keys. Table 1 summarizes information about this. In previous works [6, 8], each sender has to use not only a local message but also an aggregate-so-far tag to generate an aggregate tag, which means that we need to change the input-formats or structure of the underlying MACs. In addition, the constructions in [13, 6, 8] require other primitives except for MACs, such as a collision-resistant hash function [13], a pseudorandom permutation [6], or a pseudorandom generator [8]. On the other hand, our two constructions, SAMAC1 and SAMAC2, need not to change the input-formats or network connections of underlying MACs, and can generate an aggregate tag from MAC-tags without a secret key. While SAMAC1 requires only a MAC as a primitive, SAMAC2 needs a cryptographic hash function in addition to a MAC.

(ii) **Security.** We summarize provable security in Table 2. We note that a security proof of MT is not given, while other ones have provable security. We also note that the security proof of SAMAC2 is given in the random oracle model (ROM) while the security of HK, EFG+, and SAMAC1 are proved in the standard model (i.e., without random oracles).

(iii) **Efficiency.** Table 3 shows efficiency for the constructions. The number of function-calls, denoted by #Func.-call, shows how many times the required primitives are invoked in generating an aggregate tag. The number of function-calls in our constructions is smaller than those of the existing ones, which indicates that communication among senders and an aggregation node in our constructions is more efficient. Parallel computation in Table 3 means whether we can compute an aggregate tag in parallel. Although MT, HK, and EFG+ need to transmit an aggregate tag in a sequential way from a sender to another sender due to the order of messages, ours can compute an aggregate tag in parallel since each sender can compute a MAC-tag in parallel and then an aggregation node aggregates them into an aggregate tag following the order of messages. Parallel computability leads to less time complexity and avoids delay of sending messages in a network. Time complexity in Table 3 means the number of operations required for computing an aggregate tag. Our constructions do not need to compute MAC-tags N times owing to parallel computation of MAC-tags, while we need $(N-1)$ matrix multiplications in SAMAC1. It is not easy to strictly compare time complexity of SAMAC1 with those of MT, HK, and EFG+, since quite different operations are used. Anyway, we can say that our second construction SAMAC2 is best in time complexity since its time complexity does not depend on N. All of the constructions have the same bit-length of aggregate tags. The reduction loss being small implies that the gap between the resulting constructions and the underlying MACs in the security proof is small. From this viewpoint, SAMAC1 and SAMAC2 are superior to HK and EFG+. In total, we see that SAMAC2 is best among all ones in terms of efficiency.

In summary, in order to make the existing networks using MACs more efficient with slight change, universal applicability shown in Table 1 is important in a real world. In addition, we require provable security for the constructions, and we desire more efficiency than the current situation of the network. From the viewpoints, we consider SAMAC2 is superior to others, though the security proof is given in the random oracle model.

Furthermore, by extending the above results of SAMAC, we newly introduce a sequential aggregate MAC which has detecting functionality in this paper. Specifically, we define a model of sequential aggregate MACs that allow us to specify an invalid message or an invalid order of a certain message, which we call a *sequential aggregate MAC with detecting functionality (SAMD)*. This functionality is often useful in the case that we want to recover a valid sequence of messages in polynomial-time after we specify a sender that has sent an invalid message or a valid message with a wrong order. We also formalize the security of SAMD, called aggUF-CMA security and identifiability. Moreover, we provide a construction of SAMD: we construct a SAMD scheme in the random oracle model from any MAC scheme and any non-adaptive group testing (NGT) protocol; and we prove that the resulting SAMD meets our security definition if the underlying MAC meets UF-CMA security and the NGT is given by a d-disjunct matrix. As applications, we consider a case where a device transmits long data by data-partitioning in a wireless network. Our idea is to apply SAMD in order to reduce the numbers of tags for divided data, so that we resolve the traffic problem in a wireless network by reducing the amount of tags using our aggregation technique without changing the structure of the underlying MACs.

The rest of this paper is organized as follows: In Section 2, we briefly survey message authentication codes (MACs) and non-adaptive group testing (NGT) protocols. In Section 3, we define our model and security of sequential aggregate MACs (SAMACs), and we propose two generic constructions of

SAMAC from any MACs in Section 4. In Section 5, by extending SAMAC, we define a model and security of sequential aggregate MACs with detecting functionality (SAMD). In addition, we also provide a generic construction of SAMD from any MACs and any NGT. In Section 6, we consider the case of sending long data by data-partitioning as an application of SAMD. Finally, we conclude the paper in Section 7. The primary version of this paper appeared in [16], and this paper is an extended and full version of it. The main difference between this paper and the primary version is that this paper newly includes SAMD in Section 5 and improves Section 6 by applying SAMD rather than SAMAC.

Construction	Keyless Aggregation	Primitive	MAC's Input
MT[13]	✓	MAC and CRH	m
HK[8]		MAC and PRG	$m\|\|e\|\|\tilde{\tau}$
EFG+[6]		MAC and PRP	m
SAMAC1	✓	MAC	m
SAMAC2	✓	MAC and HF	m

Table 1: Universal Applicability: CRH means a collision-resistant hash function, PRP means a pseudorandom permutation, PRG means a pseudorandom generator, and HF is a cryptographic hash function. MAC's input means the input-format required for the underlying MAC, m is a message, e is the end-marker in a time period, and $\tilde{\tau}$ is a previous aggregate tag.

Construction	Security Level	Provable Security	Standard Model
MT[13]	Forward UF-CMA		n/a
HK[8]	Forward UF-CMA	✓	✓
EFG+[6]	UF-CMA	✓	✓
SAMAC1	aggUF-wCMA	✓	✓
SAMAC2	aggUF-CMA	✓	ROM

Table 2: Security: UF means unforgeability, UF-CMA means unforgeability against chosen message attacks, and aggUF-(w)CMA means unforgeability against (weak) chosen message attacks. The difference between aggUF(-CMA) and UF(-CMA) is that aggUF is a security which prevents an adversary from generating an aggregate-tag on at least two messages as a forgery whereas UF is a security which prevents it from generating a tag on at least one message. ROM means the random oracle model, and Standard Model means the model without any random oracles.

2 Preliminaries

In this paper, we use the following notations. For a positive integer n, let $[n] := \{1,2,\ldots,n\}$. If we write a *negligible* function ε in λ, it means a function $\varepsilon : \mathbb{N} \to [0,1]$ where $\varepsilon(\lambda) < 1/g(\lambda)$ for any polynomial g and a sufficiently large λ. We describe $\{x_i\}_{i\in[n]} := \{x_1,x_2,\ldots,x_n\}$ as a set of values x_i for all $i \in [n]$, and $(x_i)_{i\in[n]} := (x_1,x_2,\ldots,x_n)$ as a sequence of values x_i for all $i \in [n]$. For vectors $\boldsymbol{v} = (v_1,\ldots,v_n)$ and $\boldsymbol{v}' = (v'_1,\ldots,v'_n)$ in $\{0,1\}^n$, $\boldsymbol{v} \preceq \boldsymbol{v}'$ if $v_i \leq v'_i$ for every $i \in [n]$. We denote a polynomial in n by $\mathrm{poly}(n)$. Probabilistic polynomial time is abbreviated as PPT.

2.1 Message Authentication Code (MAC)

We define a deterministic message authentication code (MAC) as follows: A MAC scheme consists of three polynomial-time algorithms (KGen, Tag, Vrfy).

Construction	#Func.-call		Parallel computation	Time complexity	Agg.-tag size	Reduction loss
	Primitive	#Call				
MT[13]	MAC CRH	N N		$N \cdot T_{\mathsf{MAC}} +$ $(N-1)T_{\mathsf{CRH}}$	n	n/a
HK[8]	MAC PRG	N U		$N \cdot T_{\mathsf{MAC}} +$ $U \cdot T_{\mathsf{PRG}}$	n	$O(Nq^2)$
EFG+[6]	MAC PRP	N N		$N \cdot T_{\mathsf{MAC}} +$ $N \cdot T_{\mathsf{PRP}}$	n	$O(Nq^2L)$
SAMAC1	MAC	N	✓	$T_{\mathsf{MAC}} +$ $(N-1)T_{\mathsf{MUL}}$	n	$O(N(1-2^{-\frac{n}{4}})^{-N})$
SAMAC2	MAC HF	N 1	✓	$T_{\mathsf{MAC}} + T_{\mathsf{HF}}$	n	$O(N)$

Table 3: Efficiency: Let N be the number of senders, let q be the number of queries to the tagging oracle, let U be the number of key-updates, and let L be the maximum number of ID/message pairs in submitted queries. #Func.-call means the number that primitives are invoked in generating an aggregate tag. For a primitive $P \in \{\mathsf{MAC}, \mathsf{CRH}, \mathsf{PRG}, \mathsf{PRP}, \mathsf{HF}\}$, T_P means time complexity for computing P, and T_{MUL} means time complexity for computing multiplication of two matrices. Agg.-tag size means bit-length of aggregate tags, and n is bit-length of the underlying MAC. Reduction loss means the ratio ε/ε', where ε and ε' are the success probabilities of adversaries' attacks against the corresponding construction and the underlying MAC, respectively.

- $k \leftarrow \mathrm{KGen}(1^\lambda)$: KGen is a randomized algorithm which, on input a security parameter λ, outputs a secret key $k \in \mathscr{K}$.
- $t \leftarrow \mathrm{Tag}(k,m)$: Tag is a deterministic algorithm which, on input a secret key k and a message $m \in \mathscr{M}$, outputs a tag $t \in \mathscr{T}$.
- $1/0 \leftarrow \mathrm{Vrfy}(k,m,t)$: Vrfy is a deterministic algorithm which, on input a secret key k, a message m, and a tag t, outputs 1 (acceptance) or 0 (rejection).

Let $\mathscr{K}$ be a key-space, let $\mathscr{M}$ be a message-space, and let $\mathscr{T}$ be a tag-space. It is required that, for all $k \leftarrow \mathrm{KGen}(1^\lambda)$ and all $m \in \mathscr{M}$, we have $1 \leftarrow \mathrm{Vrfy}(k,m,\mathrm{Tag}(k,m))$.

We next define security notions of unforgeability against chosen message attacks (we call this security as UF-CMA security) and pseudorandomness for the MACs as follows:

Definition 1 (UF-CMA security)**.** *A MAC scheme* $\mathrm{MAC} = (\mathrm{KGen}, \mathrm{Tag}, \mathrm{Vrfy})$ *meets* UF-CMA *security, if for any PPT adversary* $\mathscr{A}$ *against* MAC*, the advantage of* $Adv^{uf-cma}_{MAC,\mathscr{A}}(\lambda) := \Pr[\mathscr{A}\ wins]$ *is negligible, where* $[\mathscr{A}\ wins]$ *is the event that* $\mathscr{A}$ *wins in the following game:*

Setup: *A challenger generates* $k \leftarrow \mathrm{KGen}(1^\lambda)$ *and sets* $\mathscr{L}_{Tag} = \emptyset$.

Tagging: *The tagging oracle* $\mathsf{Tag}_k(\cdot)$ *takes a query* $m \in \mathscr{M}$*, returns* $t \leftarrow \mathrm{Tag}(k,m)$*, and sets* $\mathscr{L}_{Tag} \leftarrow \mathscr{L}_{Tag} \cup \{m\}$*. The number of queries submitted by* $\mathscr{A}$ *is at most* $Q = \mathrm{poly}(\lambda)$.

Output: *When* $\mathscr{A}$ *outputs a forgery* (m^*, t^*)*,* $\mathscr{A}$ *wins if the following holds:* $1 \leftarrow \mathrm{Vrfy}(k,m^*,t^*)$*, and* $m^* \neq m$ *for any* $m \in \mathscr{L}_{Tag}$.

Definition 2 (Pseudorandomness.)**.** *A MAC scheme* $\mathrm{MAC} = (\mathrm{KGen}, \mathrm{Tag}, \mathrm{Vrfy})$ *meets pseudorandomness, if the following holds:*

$$Adv^{pr}_{MAC,\mathscr{D}}(\lambda) := \left| \Pr[\mathscr{D}^{\mathsf{Tag}_K(\cdot)}(1^\lambda) = 1] - \Pr[\mathscr{D}^{f(\cdot)}(1^\lambda) = 1] \right|$$

is negligible. Here, $\mathscr{D}$ is a PPT algorithm which, on input an oracle either $\mathsf{Tag}_k(\cdot)$ or $f(\cdot)$, determines which oracle is given; $\mathsf{Tag}_k(\cdot)$ is the tagging oracle which, on input $m \in \mathscr{M}$, returns $t = \mathrm{Tag}(k,m)$; and $f(\cdot)$ is an oracle which, on input $m \in \mathscr{M}$, returns $f(m)$ for a random function $f : \mathscr{M} \to \mathscr{T}$.

In addition, one-time pseudorandomness is defined in the same way as the pseudorandomness of MAC except that the algorithm $\mathscr{D}$ is allowed to issue at most one query.

2.2 Non-Adaptive Group Testing (NGT)

Group testing (e.g., see the book [5] for the survey) is a method to specify some special items called *defectives* among many whole items with a small number of tests than the trivial individual testing for each item. Suppose that there are totally N items of which there are d defectives in the following. The first paper about group testing is published by Dorfman [4]. The group testing techniques are classified into two types: the first type means the testing techniques by non-adaptive strategies, called non-adaptive group testing [17, 15, 7], and the second type means the techniques by adaptive strategies, called adaptive group testing (or called sequential group testing) [4, 12, 10, 7]. In non-adaptive group testing, we have to know d beforehand and we have to select all the subsets of N items to be tested without knowing the results of other tests. Finally, we try to specify all the d defectives from the results of the tests. This type of testing is typically designed by providing a d-disjunct matrix (e.g., see [5] for the survey of non-adaptive group testing). On the other hand, in adaptive group testing, we can do tests several times such that we can select a subset of items to be tested after observing the result of the previous test.

In this paper, we use non-adaptive group testing (NGT) with N items and u tests in Section 5. NGT can be represented by a matrix in $\{0,1\}^{u\times N}$, which is called a group testing matrix. For $i \in [u]$ and $j \in [N]$, the i-th test involves the item having ID id_j if and only if the (i,j) element of the corresponding group testing matrix is equal to 1. The result of a test is negative (i.e., there is no defective) if all the items involved in the test are negative, and positive (i.e., there is at least one defective) otherwise. All of the positive items (i.e., defectives) can be detected by the following procedure:

1. $J \leftarrow \{id_1, id_2 \ldots, id_N\}$, namely, J is initially the whole set consisting of all IDs of the items.
2. For $i = 1,2,\ldots,u$, do the following: If the result of the i-th test is negative, then $J \leftarrow J \backslash \{id_{i,1}, id_{i,2}, \ldots, id_{i,w_i}\}$, where $\{id_{i,1}, id_{i,2}, \ldots, id_{i,w_i}\}$ is a set of all IDs involved in the i-th test.
3. Output J

The output J includes all the defective items. It may also include the negative items in general. However, a NGT protocol can specify all d defectives among N by using the following mathematical structure called a d-disjunct matrix $\boldsymbol{G}$ as a group testing matrix:

Definition 3 (d-disjunct)**.** *A $\{0,1\}$-matrix $\boldsymbol{G}$ is said to be d-disjunct if, any d columns of $\boldsymbol{G}$ do not cover any other column of $\boldsymbol{G}$. Here, d columns $\boldsymbol{g}_{j_1}, \boldsymbol{g}_{j_2}, \ldots, \boldsymbol{g}_{j_d}$ are said to cover a column $\boldsymbol{g}$ if $\boldsymbol{g} \preceq \boldsymbol{g}_{j_1} \vee \boldsymbol{g}_{j_2} \vee \cdots \vee \boldsymbol{g}_{j_d}$ where $\vee$ is the component-wise disjunction.*

3 Sequential Aggregate MACs: Our Model and Security

We introduce a new model of sequential aggregate MACs (SAMACs) where an aggregation algorithm generates a sequential aggregate tag depending only on multiple and independent MAC tags without any secret-key, and we formally define security in this model.

Let MAC=(KGen, Tag, Vrfy) be a MAC scheme. Then, a sequential aggregate MAC (SAMAC) scheme consists of a tuple of five polynomial-time algorithms (KGen, Tag, Vrfy, SeqAgg, SAVrfy) as

follows, where N is the number of senders, ID is an ID-space, $\mathscr{K}$ is a key-space, $\mathscr{M}$ is a message-space, $\mathscr{T}$ is a tag-space, and $\mathscr{T}_{agg}$ is an aggregate tag-space. Let $\mathscr{S} := \{(id_{\ell_1}, id_{\ell_2}, \ldots, id_{\ell_{\widehat{N}}}) \mid \widehat{N} \leq N \wedge id_i \neq id_j \text{ if } i \neq j\}$, which means the set of all different sequences of IDs with length at most N:

- $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$: KGen is a randomized algorithm which, on input a security parameter λ and an ID $id \in \mathsf{ID}$, outputs a secret key $k_{id} \in \mathscr{K}$. Note that this is the same as KGen of the underlying MAC except for adding id.
- $t \leftarrow \mathrm{Tag}(k_{id}, m)$: Tag is a deterministic algorithm which, on input a secret key k_{id} and a message m, outputs a tag $t \in \mathscr{T}$. This is the same as Tag of the MAC.
- $1/0 \leftarrow \mathrm{Vrfy}(k_{id}, m, t)$: Vrfy is a deterministic algorithm which, on input a secret key k_{id}, a message $m \in \mathscr{M}$, and a tag t, outputs 1 (acceptance) or 0 (rejection). This is the same as Vrfy of the MAC.
- $\tau \leftarrow \mathrm{SeqAgg}(T)$: SeqAgg is a deterministic algorithm which, on input a sequence of tags $T = ((id_{\ell_i}, t_i))_{i \in [\widehat{N}]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathscr{S}$, outputs an aggregate tag $\tau \in \mathscr{T}_{agg}$.
- $1/0 \leftarrow \mathrm{SAVrfy}(K, M, \tau)$: SAVrfy is a deterministic algorithm which, on input a set of key/id pairs $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$, a sequence of message/id pairs $((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$ for any $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathscr{S}$, and an aggregate tag τ, outputs 1 (acceptance) or 0 (rejection).

We require that the following condition (i.e., correctness) holds:

- For all $id \in \mathsf{ID}$, all $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$ and all $m \in \mathscr{M}$, we have $1 \leftarrow \mathrm{Vrfy}(k_{id}, m, \mathrm{Tag}(k_{id}, m))$.
- For all $id \in \mathsf{ID}$, all $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$ and all $m \in \mathscr{M}$, for any $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$ and any $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathscr{S}$, we have $1 \leftarrow \mathrm{SAVrfy}(K, M, \tau)$, where $T = ((id_{\ell_i}, \mathrm{Tag}(k_{id_{\ell_i}}, m_i)))_{i \in [\widehat{N}]}$ and $\tau = \mathrm{SeqAgg}(T)$.

We next define a security notion of *C-aggregate unforgeability against chosen message attacks* (we call this security as C-aggUF-CMA security) in our model.

Definition 4 (C-aggUF-CMA security)**.** *A sequential aggregate MAC scheme* SAMAC = (KGen, Tag, Vrfy, SeqAgg, SAVrfy) *meets C-aggUF-CMA security, if for any PPT adversary $\mathscr{A}$ against* SAMAC*, the advantage $Adv^{\text{agg-uf}}_{\text{SAMAC},\mathscr{A}}(\lambda) := \Pr[\mathscr{A} \text{ wins}]$ of $\mathscr{A}$ is negligible, where $[\mathscr{A} \text{ wins}]$ is the event that $\mathscr{A}$ wins in the following game:*

Setup: *A challenger generates a set of secret-key/ID pairs $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$ by using the KGen algorithm. Then, it sets lists $\mathscr{L}_{Cor} = \emptyset$ and $\mathscr{L}_{SA} = \emptyset$.*

Corrupt: *The corrupt oracle $\mathsf{Corrupt}(\cdot)$ takes an ID $id \in \mathsf{ID}$ as input and returns the secret key k_{id} and sets $\mathscr{L}_{Cor} \leftarrow \mathscr{L}_{Cor} \cup \{id\}$, where $\mathscr{L}_{Cor}$ means a list of IDs whose corresponding secret keys are known by an adversary. The number of queries submitted by $\mathscr{A}$ is at most C.*

Tagging: *The sequential aggregate tagging oracle $\mathsf{SATag}_K(\cdot)$ takes a sequence of message/ID pairs $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_{\widehat{N}}}) \in \mathscr{S}$ and the number of not corrupted IDs is at least $N - C$. Then, it does the following:*

1. *Compute $t_i \leftarrow \mathrm{Tag}(k_{id_{\ell_i}}, m_i)$ for all $i \in [\widehat{N}]$,*
2. *Output $\tau \leftarrow \mathrm{SeqAgg}(((id_{\ell_i}, t_i))_{i \in [\widehat{N}]})$,*
3. *Set $\mathscr{L}_{SA} \leftarrow \mathscr{L}_{SA} \cup \{M\}$.*

The number of queries which $\mathscr{A}$ submits is at most $Q = \mathrm{poly}(\lambda)$. $\mathscr{A}$ is not allowed to access $\mathsf{Corrupt}(\cdot)$ after accessing $\mathsf{SATag}_K(\cdot)$.

Output: *When $\mathscr{A}$ outputs $M^* = ((m_i^*, id_{\ell_i^*}))_{i\in[\widetilde{N}]}$ and τ^* ($\widetilde{N} \le N$), $\mathscr{A}$ wins if the following holds:*

- $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$,
- $(id_{\ell_1^*}, \ldots, id_{\ell_{\widetilde{N}}^*}) \in \mathscr{S}$ *such that the number of not-corrupted IDs is at least $N - C$,*
- $M^* \notin \mathscr{L}_{SA}$ *such that* $id_{\ell_1^*}, id_{\ell_{\widetilde{N}}^*} \notin \mathscr{L}_{Cor}$.

In addition, we define a weak variant of C-aggUF-CMA security (called C-aggUF-wCMA security): The definition of C-aggUF-wCMA security is the same as that of C-aggUF-CMA security except for the following conditions:

- *For each query to the oracle $\mathsf{SATag}_K(\cdot)$, the number of message/ID pairs is always N (i.e., $\widehat{N} = N$), and the output of $\mathscr{A}$ meets $\widetilde{N} = N$.*
- *$\mathscr{A}$ is allowed to issue only queries $((id_{\ell_i}, m_i))_{i\in[N]}$ to $\mathsf{SATag}_K(\cdot)$ such that any pair of a message and not-corrupted ID among $\{(id_{\ell_i}, m_i)\}_{i\in[N]}$ has never been included in any previous sequences queried to $\mathsf{SATag}_K(\cdot)$. Namely, for each query to $\mathsf{SATag}_K(\cdot)$, $\mathscr{A}$ cannot issue queries $((id_{\ell_i}, m_i))_{i\in[N]}$ such that there exists $id_{\ell_i} \notin \mathscr{L}_{Cor}$ and (id_{ℓ_i}, m_i) is included in some prior ID/message-sequence query $M \in \mathscr{L}_{SA}$.*

In addition, in principle, it is impossible in our model to guarantee the unforgeability against an adversary who can observe each MAC-tag before the aggregation. This reason is that, if the adversary obtains a sequence $\{(m_i, \mathsf{Tag}(k_{id_i}, m_i))\}_{i\in[\widehat{N}]}$ by accessing the tagging oracle which returns a tag $\mathrm{Tag}(k, m_i)$ for a query m_i, he can generate an aggregate tag for any sequential messages $(m_{\ell_i})_{i\in[\widehat{N}]}$ because SeqAgg algorithm is keyless. Thus, we consider the attacking model where an adversary makes a forgery by only accessing the sequential aggregate tagging oracle $\mathsf{SATag}_K(\cdot)$.

We next show a condition for a SeqAgg algorithm to achieve C-aggUF-CMA security. For simplicity, we view a SeqAgg algorithm as a function $F : \mathscr{T}^N \to \mathscr{T}_{agg}$, where $\mathscr{T}$ is a MAC tag-space and $\mathscr{T}_{agg}$ is an aggregate tag-space. Then, the following proposition states that, for given $y \in \mathscr{T}_{agg}$, the equation $F(x) = y$ should not be correctly solved in polynomial time for achieving C-aggUF-CMA.

Proposition 1. *Let* SAMAC = (KGen, Tag, Vrfy, SeqAgg, SAVrfy) *be a sequential aggregate MAC scheme, and we identify SeqAgg with $F : \mathscr{T}^N \to \mathscr{T}_{agg}$. If we can compute a sequence of tags $(t_i)_{i\in[N]}$ from $F((t_i)_{i\in[N]})$ in polynomial time, then SAMAC does not meet C-aggUF-CMA security.*

Proof. Let $\mathscr{A}$ be a PPT adversary against SAMAC. If $\mathscr{A}$ gets an aggregate tag τ on a sequence of messages $(m_i)_{i\in[N]}$ by accessing the sequential aggregate tagging oracle, $\mathscr{A}$ can compute each MAC-tag t_i on m_i for all $i \in [N]$. For a message sequence $(m_{\ell_i})_{i\in[N]}$ different from $(m_i)_{i\in[N]}$ such that $\{m_{\ell_i}\}_{i\in[N]} = \{m_i\}_{i\in[N]}$, $\mathscr{A}$ can make a forgery $\tau^* = F((t_{\ell_i})_{i\in[N]})$ on $(m_{\ell_i})_{i\in[N]}$. □

4 Construction of Sequential Aggregate MACs

4.1 Our Construction

We propose a generic construction of sequential aggregate MACs (SAMACs) in our model such that our SAMAC consists of any MAC scheme MAC = (MAC.KGen, MAC.Tag, MAC.Vrfy) and a sequential aggregation algorithm-pair (SA.SeqAgg, SA.SAVrfy). This is well explained by the following construction of sequential aggregate MACs, GSAMAC = (KGen, Tag, Vrfy, SeqAgg, SAVrfy):

- $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$: Generate $k_{id} \leftarrow \mathrm{MAC.KGen}(1^\lambda)$ and output k_{id}.
- $t \leftarrow \mathrm{Tag}(k_{id}, m)$: Output a MAC-tag $t \leftarrow \mathrm{MAC.Tag}(k_{id}, m)$.
- $1/0 \leftarrow \mathrm{Vrfy}(k_{id}, m, t)$: Output $b \leftarrow \mathrm{MAC.Vrfy}(k_{id}, m, t) \in \{0,1\}$.
- $\tau \leftarrow \mathrm{SeqAgg}(T)$: Take a sequence of MAC tags $T = ((id_{\ell_i}, t_i))_{i \in [\widehat{N}]}$ as input and output $\tau \leftarrow \mathrm{SA.SeqAgg}(T)$.
- $1/0 \leftarrow \mathrm{SAVrfy}(K, M, \tau)$: Take $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$, $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$, and τ as input, and output a bit $b \leftarrow \mathrm{SA.SAVrfy}(K, M, \tau)$.

Therefore, it is enough to construct only a sequential aggregation algorithm-pair (SA.SeqAgg, SA.SAVrfy), and we propose two constructions called SA1 and SA2 for it. Consequently, we will obtain two constructions of SAMACs called SAMACi ($i = 1, 2$) starting from any MAC scheme and the aggregate algorithm-pair SAi.

We construct two aggregation algorithm-pairs SA1 and SA2. First, we describe the basic processes of SA1 informally as follows.

- SeqAgg algorithm:
  1. Each MAC-tag t_i is transformed into a matrix T_i,
  2. Output the product of these matrices $T_1 T_2 \cdots T_{\widehat{N}}$ as an aggregate tag τ.
- SAVrfy algorithm:
  1. Compute an aggregate tag τ' from $K = \{(k_{id_i}, id_i)\}_{i \in [N]}$ and $M = ((m_i, id_{\ell_i}))_{i \in [\widehat{N}]}$ following the SeqAgg algorithm,
  2. Output 1 (accept) if $\tau' = \tau$, or output 0 (reject) otherwise.

Our idea is based on non-commutativity of matrix multiplications. And, the order of messages is regarded as invalid, if the order of MAC-tags' matrices are changed. From this, we can construct SAMAC schemes from any MAC schemes by transforming each MAC-tag into a matrix, and can give a security proof in the standard model. Also, we provide a simple construction SA2 by using hash functions, and give a security proof in the random oracle model.

Besides, it should be noted that we cannot achieve the security under consideration even if we slightly change inputs of the underlying MACs as follows: Suppose that, for a sequence $(id_{\ell_1}, \ldots, id_{\ell_N}) \in \mathscr{S}$, each sender having ID id_i computes $t_i \leftarrow \mathrm{Tag}(k_{id_i}, id_i \parallel m_i)$ and the resulting aggregate tag is $\tau = t_1 \oplus \cdots \oplus t_N$. However, it is easy to generate a valid forgery in the case where some IDs are corrupted. Actually, for an aggregate tag $\tau = t_1 \oplus \cdots \oplus t_N$ on $(m_i)_{i \in [N]}$, an adversary can compute $t_i' \leftarrow \mathrm{Tag}(k_{id_i}, id_i \parallel m_i')$ with a corrupted ID's (i.e., id_i) secret key k_{id_i} and can generate a forgery $\tau' = \tau \oplus t_i \oplus t_i'$ without accessing the tagging oracle. Furthermore, even if an adversary does not corrupt any secret keys, he can break aggUF-(w)CMA security by submitting queries to the tagging oracle and receiving the following aggregate-tags: For $m_1' \neq m_1$ and $m_2' \neq m_2$,

$$\begin{aligned}
\tau_1 &= \mathrm{Tag}(k_{id_1}, id_1 \parallel m_1) \oplus \mathrm{Tag}(k_{id_2}, id_2 \parallel m_2) \oplus \mathrm{Tag}(k_{id_3}, id_3 \parallel m_3) \oplus \cdots, \\
\tau_2 &= \mathrm{Tag}(k_{id_1}, id_1 \parallel m_1') \oplus \mathrm{Tag}(k_{id_2}, id_2 \parallel m_2) \oplus \mathrm{Tag}(k_{id_3}, id_3 \parallel m_3) \oplus \cdots, \\
\tau_3 &= \mathrm{Tag}(k_{id_1}, id_1 \parallel m_1) \oplus \mathrm{Tag}(k_{id_2}, id_2 \parallel m_2') \oplus \mathrm{Tag}(k_{id_3}, id_3 \parallel m_3) \oplus \cdots.
\end{aligned}$$

Then, he computes $\tau_1 \oplus \tau_2 \oplus \tau_3 = \mathrm{Tag}(k_{id_1}, id_1 \parallel m_1') \oplus \mathrm{Tag}(k_{id_2}, id_2 \parallel m_2') \oplus \mathrm{Tag}(k_{id_3}, id_3 \parallel m_3) \oplus \cdots$, which is a valid forgery since the sequence $((m_1', id_1), (m_2', id_2), (m_3, id_3), \ldots)$ has never been queried. Therefore, this construction does not meet the security of sequential aggregate MACs in our model.

Construction 1. We propose a construction SA1 by transforming each MAC-tag t_i into a matrix as follows: Let n be bit-length of MAC-tag and we separate a MAC-tag $t_i \in \{0,1\}^n$ into $(t_{i,1} \| t_{i,2} \| t_{i,3} \| t_{i,4}) \in (\{0,1\}^{\frac{n}{4}})^4$. Then, we regard each $t_{i,j} \in \{0,1\}^{\frac{n}{4}}$ $(1 \leq j \leq 4)$ as an element of the finite field $GF(2^{\frac{n}{4}})$, and set $T_i := \begin{bmatrix} t_{i,1} & t_{i,2} \\ t_{i,3} & t_{i,4} \end{bmatrix}$. Here, we note that such a matrix T_i is invertible with an overwhelming probability if the MAC meets pseudorandomness. Based on this transformation, SA1 = (SA1.SeqAgg, SA1.SAVrfy) is constructed in the following way: The number of aggregated MAC tags is fixed, namely N ($N \geq 2$).

- $\tau \leftarrow \text{SA1.SeqAgg}(((id_{\ell_i}, t_i))_{i\in[N]})$: Generate an aggregate tag as follows:
  1. For each $i \in [N]$, let $T_i := \begin{bmatrix} t_{i,1} & t_{i,2} \\ t_{i,3} & t_{i,4} \end{bmatrix}$ be a matrix transformed from t_i as mentioned above.
  2. Output $\tau := T_1 T_2 \cdots T_N$.
- $1/0 \leftarrow \text{SA1.SAVrfy}(K, M, \tau)$: For $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ and $M = ((m_i, id_{\ell_i}))_{i\in[N]}$, verify (M, τ) as follows:
  1. For each $i \in [N]$, compute $t'_i \leftarrow \text{MAC.Tag}(k_{id_{\ell_i}}, m_i)$ and $\tau' \leftarrow \text{SA1.SeqAgg}(((id_{\ell_i}, t'_i))_{i\in[N]})$.
  2. Output 1 if $\tau' = \tau$, or output 0 otherwise.

Then, we show the following lemma.

Lemma 1. *Given two aggregate tags* $\tau_1 \leftarrow \text{SA1.SeqAgg}((id_{\ell_1}, t_1), \ldots, (id_{\ell_i}, t_i))$ *and* $\tau_2 \leftarrow \text{SA1.SeqAgg}((id_{\ell_{i+1}}, t_{i+1}), \ldots, (id_{\ell_j}, t_j))$*, if MAC meets pseudorandomness, the probability that* $\tau_1\tau_2 = \tau_2\tau_1$ *holds is negligible. More generally, for* $s \geq 2$, $\tau_1, \tau_2, \ldots, \tau_s \xleftarrow{U} GF(2^{\frac{n}{4}})^{2\times 2}$*, the probability that* $\tau_1\tau_2\cdots\tau_s = \tau_{\sigma(1)}\tau_{\sigma(2)}\cdots\tau_{\sigma(s)}$ *holds is negligible for any permutation* $\sigma \neq 1$ *over* $\{1,2,\ldots,s\}$.

Proof. We denote the matrices τ_1 and τ_2 by

$$\tau_1 = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix},\ \tau_2 = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix},$$

where $a_i, b_i, c_i, d_i \in GF(2^{\frac{n}{4}})$ for $i \in \{1,2\}$. We have

$$\tau_1\tau_2 = \begin{bmatrix} a_1a_2 + b_1c_2 & a_1b_2 + b_1d_2 \\ a_2c_1 + c_2d_1 & b_2c_1 + d_1d_2 \end{bmatrix},\ \tau_2\tau_1 = \begin{bmatrix} a_1a_2 + b_2c_1 & a_2b_1 + b_2d_1 \\ a_1c_2 + c_1d_2 & b_1c_2 + d_1d_2 \end{bmatrix}.$$

Then, $\tau_1\tau_2 = \tau_2\tau_1$ is equivalent to the conditions:

$$b_1c_2 = b_2c_1, \quad (1)$$
$$a_1b_2 + b_1d_2 = a_2b_1 + b_2d_1, \quad (2)$$
$$a_2c_1 + c_2d_1 = a_1c_2 + c_1d_2. \quad (3)$$

Hence, the number of the equations that must hold is three, while the number of variables a_i, b_i, c_i, d_i $(i = 1,2)$ is eight. Therefore, if a_i, b_i, c_i, d_i $(i = 1,2)$ are chosen uniformly at random, the probability that the equations (1)-(3) hold is $(2^{-\frac{n}{4}})^3 = 2^{-\frac{3}{4}n}$. Therefore, if the MAC meets pseudorandomness, the probability that the equations (1)-(3) hold is negligible. More generally, by taking into account the number of variables and that of equations in $\tau_1\tau_2\cdots\tau_s = \tau_{\sigma(1)}\tau_{\sigma(2)}\cdots\tau_{\sigma(s)}$, the probability that $\tau_1\tau_2\cdots\tau_s = \tau_{\sigma(1)}\tau_{\sigma(2)}\cdots\tau_{\sigma(s)}$ holds is $O(2^{-\frac{1}{4}n})$. □

Construction 2. We construct SA2 = (SA2.SeqAgg, SA2.SAVrfy) by using hash functions in a simple way, and this construction is provably secure in the random oracle model. SA2 is given as follows: Let H be a hash function $H : \{0,1\}^* \to \mathscr{T}$, where $\mathscr{T}$ is the tag space of a MAC scheme.

- $\tau \leftarrow \mathrm{SA2.SeqAgg}(((id_{\ell_i}, t_i))_{i\in[\widehat{N}]})$: Output $\tau := H(t_1, \ldots, t_{\widehat{N}})$.
- $1/0 \leftarrow \mathrm{SA2.SAVrfy}(K, M, \tau)$: Output 1 if $\tau = H(t'_1, \ldots, t'_{\widehat{N}})$, where $t'_i := \mathrm{MAC.Tag}(k_{id_{\ell_i}}, m_i)$ for all $i \in [\widehat{N}]$, and output 0 otherwise.

By definition of hash functions, we can see that the order of messages is guaranteed.

4.2 Security of Our Constructions

The following theorems show the security of our constructions.

Theorem 1. *If MAC meets pseudorandomness, SAMAC1 meets* 0*-aggUF-wCMA security.*

Proof. Let $\mathscr{A}$ be a PPT adversary against SAMAC1. We consider the case where the number of queries issued to Corrupt oracle is $C = 0$. We define the following events:

- Succ: An event that $\mathscr{A}$ outputs a forgery breaking aggUF-CMA.
- New: An event that for a new message which is never queried, $\mathscr{A}$ makes a forgery against a MAC scheme.
- Pre: An event that $\mathscr{A}$ makes a forgery against SAMAC1 without generating any forgeries against MACs.
- Replace: An event that $\mathscr{A}$ makes a forgery by changing the order of a sequence of message/ID pairs that was queried to the sequential aggregate tagging oracle.

Because the events New and Pre are exclusive, we have

$$\begin{aligned} Adv^{agg-uf}_{SAMAC,\mathscr{A}}(\lambda) &:= \Pr[\mathsf{Succ}] \\ &= \Pr[\mathsf{Succ} \wedge \mathsf{New}] + \Pr[\mathsf{Succ} \wedge \mathsf{Pre}] \\ &= \Pr[\mathsf{Succ} \wedge \mathsf{New}] + \Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}] + \Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Replace}}]. \end{aligned}$$

Therefore, it is sufficient to prove the following:

- $\Pr[\mathsf{Succ} \wedge \mathsf{New}] \leq \frac{N}{P_{inv}} Adv^{pr}_{MAC,\mathscr{F}}(\lambda) + \delta(\lambda)$ for a function P_{inv} of N and a negligible function δ.
- $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}] \leq \varepsilon(\lambda)$ for a negligible function ε.
- $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Replace}}] \leq \frac{N}{P_{inv}} \cdot Adv^{pr}_{MAC,\mathscr{D}}(\lambda) + \frac{1}{2^n}$.

Event $[\mathsf{Succ} \wedge \mathsf{New}]$: In this case, an adversary generates a forgery against a MAC scheme. By using $\mathscr{A}$ breaking aggUF-wCMA security, we construct a PPT algorithm $\mathscr{F}$ breaking the UF-CMA security of MACs as follows.

Setup: Given the tagging oracle of a MAC, do the following.

1. Generate $k_{id_i} \leftarrow \mathrm{KGen}(1^\lambda, id_i)$ for all $i \in [N]$,
2. Set a list $\mathscr{L}_{SA} = \emptyset$.

3. Choose an ID $id^* \in \mathsf{ID}$ uniformly at random.

Tagging: For each query $M = ((m_i, id_{\ell_i}))_{i\in[N]}$ to the oracle $\mathsf{SATag}_K(\cdot)$, do the following for all $i \in [N]$ where $K := \{(k_{id_i}, id_i)\}_{i\in[N]}$.

- If $id_{\ell_i} \neq id^*$, compute $t_i = \mathrm{Tag}(k_{id_{\ell_i}}, m_i)$,
- If $id_{\ell_i} = id^*$, submit a message query m_i to the MAC oracle and receive the tag t_i.

Return $\tau = \mathrm{SeqAgg}((id_{\ell_i}, t_i)_{i\in[N]})$ to $\mathscr{A}$ and set $\mathscr{L}_{SA} \leftarrow \mathscr{L}_{SA} \cup \{M\}$.

Output: When $\mathscr{A}$ outputs $M^* = ((m_i^*, id_{\ell_i^*}))_{i\in[N]}$ and τ^*, do the following.

1. Move to the next step if the output of $\mathscr{A}$ meets the conditions of the security game except for $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$, or abort this game otherwise.
2. For $i \in [N]$ and id_{ℓ_i} except for id^*, compute $t_i^* = \mathrm{MAC.Tag}(k_{id_{\ell_i^*}}, m_i^*)$.
3. Let i^* be the order of the ID id^* in M^* and compute T_{i^*} in the following way: $T_{i^*} = (T_{i^*-1}^*)^{-1} \cdot \cdots \cdot (T_1^*)^{-1} \cdot \tau \cdot (T_N^*)^{-1} \cdot \cdots \cdot (T_{i^*+1}^*)^{-1} \in GF(2^{n/4})^{2\times 2}$.
4. Recover a MAC-tag t_{i^*} from the matrix T_{i^*}.
5. Output (m_{i^*}, t_{i^*}) as a forgery of the MAC.

$\mathscr{F}$ simulates the environment of $\mathscr{A}$ completely. In Step 3 of Output phase, the probability that all matrices are invertible is at least $P_{inv} := (1 - \frac{1}{2^{n/4}})^{N-1}$. For all IDs $id_{\ell_i^*}$ ($i \in [N]$) except for id^*, MAC tags t_i^* generated by using $k_{id_{\ell_i^*}}$ are valid. Therefore, t_{i^*} is also a valid MAC tag. Therefore, the success probability of $\mathscr{F}$ is at least $\frac{P_{inv}}{N} \cdot \Pr[\mathsf{Succ} \wedge \mathsf{New}]$, and hence $Adv^{uf-cma}_{\mathrm{MAC},\mathscr{F}}(\lambda) \geq \frac{P_{inv}}{N} \cdot \Pr[\mathsf{Succ} \wedge \mathsf{New}]$. By [3], we have $\Pr[\mathsf{Succ} \wedge \mathsf{New}] \leq \frac{N}{P_{inv}} \left(Adv^{pr}_{\mathrm{MAC},\mathscr{F}}(\lambda) + \frac{1}{2^n} \right)$ and hence $\Pr[\mathsf{Succ} \wedge \mathsf{New}] \leq \frac{N}{P_{inv}} Adv^{pr}_{\mathrm{MAC},\mathscr{F}}(\lambda) + \delta(\lambda)$.

Event $[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}]$: From Lemma 1, we have $\Pr[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \mathsf{Replace}] \leq \varepsilon(\lambda)$.

Event $[\mathsf{Succ} \wedge \mathsf{Pre} \wedge \overline{\mathsf{Replace}}]$: In this event, we consider the following adversaries: We assume that for a query M, there exists a pair (id, m, t) between not corrupted pairs $(id_{i_1}, m_{i_1}, t_{i_1})$ and $(id_{i_N}, m_{i_N}, t_{i_N})$ such that i_1 is the first order among all IDs and i_N is the last order among all IDs, in M. An adversary tries to replace the message/tag pair (m,t) with (m^*, t^*) such that $m^* \neq m$ and $t^* = \mathsf{Tag}(k_{id}, m^*)$ in the similar way as the attack against the construction computing XOR of all tags, which we explained in Section 4.1. However, he cannot replace the message/tag pair without knowing t_{i_1} and t_{i_N}. We show that the probability that the event happens is negligible if MACs meet pseudorandomness.

Let Game-0 be the standard security game and let $C = 0$ be the number of corrupted IDs. For $X \in [N]$, we define Game-X where for one of IDs $id \in \mathsf{ID}$, the MAC's tagging algorithm is replaced with a random function $f_{id} : \mathscr{M} \to \mathscr{T}$. Then, we show that in Game-$(X-1)$ and Game-X, the difference between the success probabilities of them is negligible from pseudorandomness of MACs. We construct a PPT algorithm $\mathscr{D}$ breaking pseudorandomness of a MAC scheme. $\mathscr{D}$ can be constructed in the same way as in the above $\mathscr{F}$ except for the process of Output phase. We describe the process of $\mathscr{D}$ at Output phase as follows: When $\mathscr{A}$ outputs $M^* = ((m_i^*, id_{\ell_i^*}))_{i\in[N]}$ and τ^*, do the following.

1. Move to the next step if the output of $\mathscr{A}$ meets the conditions of the security game except for $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$, or abort this game otherwise.
2. For each $id_{\ell_i^*}$ ($i \in [N]$) except for id^*, compute t_i^* by using the key $k_{id_{\ell_i^*}}$.
3. Compute the MAC tag t_{id^*} of id^* from τ^* and the other tags as computed in Step 2.

4. Submit m^* to the tagging oracle of MAC and receive the tag t.
5. Output 1 if $t_{id^*} = t$ and (m^*, id^*) has never been queried, or output 0 otherwise.

In Game-N, all outputs of f_{id} and MAC tags are hidden statistically. Therefore, the probability is at most $\frac{N}{P_{inv}} \cdot Adv^{pr}_{MAC,\mathscr{D}}(\lambda) + \frac{1}{2^n}$.

From the discussion above, we have

$$Adv^{agg-uf}_{SAMAC1,\mathscr{A}}(\lambda) \leq \frac{2N}{P_{inv}} \cdot Adv^{pr}_{MAC}(\lambda) + \frac{1}{2^n} + \delta(\lambda) + \varepsilon(\lambda),$$

where $P_{inv} = \left(1 - \frac{1}{2^{n/4}}\right)^{N-1}$ Therefore, the proof is completed. □

Theorem 2. *If MAC meets UF-CMA, SAMAC2 meets $(N-2)$-aggUF-CMA security.*

Proof. Let $\mathscr{A}$ be a PPT adversary against SAMAC2. Let $Q_h = \mathrm{poly}(\lambda)$ be the number of queries submitted by $\mathscr{A}$ to the random oracle $H(\cdot)$, and let $Q_t = \mathrm{poly}(\lambda)$ be the number of queries issued to the tagging oracle $\mathsf{SATag}_K(\cdot)$. Let $\mathscr{L}_H$ be the list of the query/answer pairs of $H(\cdot)$. Let Forge be an event that $\mathscr{A}$ breaks SAMAC2 by making a forgery of the underlying MAC, and let Coll be an event that $\mathscr{A}$ finds a collision of the random oracle H. Then, we have $Adv^{agg-uf}_{SAMAC2,\mathscr{A}}(\lambda) := \Pr[\mathsf{Forge}] \leq \Pr[\mathsf{Coll}] + \Pr[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}]$.

In the event Coll, an adversary tries to find a collision of H. We note that this case includes an attack that he replaces MAC-tags queried to H. The success probability is at most $\frac{(Q_h+Q_t)^2}{2^{n+1}}$ because the number of accessing the random oracle is at most $(Q_h + Q_t)$.

Next, we consider the event $[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}]$. We construct a PPT algorithm $\mathscr{F}$ breaking the UF-CMA security of a MAC scheme in the following way:

Setup: Given the tagging oracle of MAC, do the following.

1. Generate $k_{id_i} \leftarrow \mathrm{KGen}(1^\lambda, id_i)$ for all $i \in [N]$,
2. Set lists $\mathscr{L}_{Cor} = \emptyset$, $\mathscr{L}_{SA} = \emptyset$, and $\mathscr{L}_H = \emptyset$.

Corrupt: When $\mathscr{A}$ submits an ID $id \in \mathsf{ID}$ to the oracle $\mathsf{Corrupt}(\cdot)$, return k_{id} and set $\mathscr{L}_{Cor} \leftarrow \mathscr{L}_{Cor} \cup \{id\}$. When $\mathscr{A}$ stops accessing $\mathsf{Corrupt}$ and moves to the $\mathsf{Tagging}$ phase, choose an ID $id^* \notin \mathscr{L}_{Cor}$ uniformly at random.

Tagging: For each query $M = ((m_i, id_{\ell_i}))_{i\in[N]}$ to the oracle $\mathsf{SATag}_K(\cdot)$, where $K := \{(k_{id_i}, id_i)\}_{i\in[N]}$, do the following for all $i \in [N]$.

- If $id_{\ell_i} \neq id^*$, compute $t_i = \mathrm{Tag}(k_{id_{\ell_i}}, m_i)$,
- If $id_{\ell_i} = id^*$, submit a message query m_i to the MAC oracle and receive the tag t_i.

Return $\tau = \mathrm{SeqAgg}((id_{\ell_i}, t_i)_{i\in[N]})$ to $\mathscr{A}$ and set $\mathscr{L}_{SA} \leftarrow \mathscr{L}_{SA} \cup \{M\}$.

And also, for each query $(t_1, \ldots, t_{\widehat{N}})$ to the random oracle H, do the following.

1. Search for a pair $((t_1, \ldots, t_{\widehat{N}}), \tau) \in \mathscr{L}_H$.
2. If there exists such a pair, return τ.
3. Otherwise, return $\tau \xleftarrow{U} \mathscr{T}$ and set $\mathscr{L}_H \leftarrow \mathscr{L}_H \cup \{((t_1, \ldots, t_{\widehat{N}}), \tau)\}$.

Output: When $\mathscr{A}$ outputs $M^* = ((m^*_i, id_{\ell^*_i}))_{i\in[N]}$ and τ^*, do the following.

1. Move to the next step if the output of $\mathscr{A}$ meets the conditions of the security game except for $1 \leftarrow \mathrm{SAVrfy}(K, M^*, \tau^*)$, or abort this game otherwise.
2. Compute $t_i^* = \mathrm{MAC.Tag}(k_{id_{\ell_i^*}}, m_i)$ except for id^*,
3. Find a pair $((t_i^*)_{i\in[\widetilde{N}]}, \tau^*)$ except for a tag of id^* from $\mathscr{L}_H$. Abort this game if there exists no such pair in $\mathscr{L}_H$.
4. Output the id^*'s pair (m^*, t^*).

The pair $(t_1^*, \ldots, t_{\widetilde{N}}^*, \tau^*)$ is in $\mathscr{L}_H$ with overwhelming probability because the probability that it outputs τ^* such that $\tau^* = H(t_1^*, \ldots, t_{\widetilde{N}}^*)$ is negligible without accessing the random oracle $H(\cdot)$. Thus, $\mathscr{F}$'s output is a valid forgery breaking a MAC scheme. Therefore, we have $\Pr[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}] \leq N \cdot Adv^{uf-cma}_{MAC,\mathscr{F}}(\lambda)$.

From the above discussion, we obtain

$$Adv^{agg-uf}_{SAMAC2,\mathscr{A}}(\lambda) \leq N \cdot Adv^{uf}_{MAC,\mathscr{F}}(\lambda) + \frac{(Q_h + Q_t)^2}{2^{n+1}}.$$

Therefore, the proof is completed. □

5 Sequential Aggregate MACs with Detecting Functionality

In this section, we introduce a new model of sequential aggregate MACs which have more advanced functionality: A sequential aggregate MAC allows us to specify an invalid message or an invalid order of a certain message, while the verification algorithm SAVrfy of SAMAC in Sections 3 and 4 only outputs 1 (i.e., both all the messages and their order are valid) or 0 (i.e., otherwise). We call such sequential aggregate MACs *sequential aggregate MACs with detecting functionality (SAMD)*. This functionality is often useful in the case that we want to recover a valid sequence of messages in polynomial-time after we specify a sender that has sent an invalid message or a valid message with a wrong order.

We define a model and security of SAMD in Section 5.1, and we provide a construction of SAMD in Section 5.2. In our construction, we utilize an SAMAC in Section 4 and non-adaptive group testing (NGT). A NGT protocol with a group testing matrix can detect a specific items among a lot of items (see Section 2.2). From the properties of NGT protocols and keyless sequential aggregate MACs, we can detect an invalid ID whose message or its order is wrong. Although the security notion of (non-sequential) aggregate MACs with detecting functionality was formalized in [9], we also need to consider integrity of the order of messages in addition to the security notion in [9]. Therefore, we formalize a model and security of SAMD, and then propose a construction that meets our security definition in this section.

5.1 Our Model and Security of SAMD

A sequential aggregate MAC with detecting functionality (SAMD) consists of five polynomial-time algorithms (KGen, Tag, Vrfy, DSeqAgg, DSAVrfy) as follows: N is the number of senders, ID is an ID-space, $\mathscr{K}$ is a key-space, $\mathscr{M}$ is a message-space, $\mathscr{T}$ is a tag-space, and $\mathscr{T}_{agg} = \mathscr{T}^u$ is an aggregate tag-space, where u is the number of tags. Let $\mathscr{S} := \{(id_{\ell_1}, id_{\ell_2}, \ldots, id_{\ell_N}) \mid id_i \neq id_j \text{ if } i \neq j\}$, which means the set of all different sequences of IDs with length N:

- $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$: KGen is a randomized algorithm which, on input a security parameter λ and an ID $id \in \mathsf{ID}$, outputs a secret key $k_{id} \in \mathscr{K}$.
- $t \leftarrow \mathrm{Tag}(k_{id}, m)$: Tag is a deterministic algorithm which, on input a secret key k_{id} and a message m, outputs a tag $t \in \mathscr{T}$.

- $1/0 \leftarrow \mathrm{Vrfy}(k_{id}, m, t)$: Vrfy is a deterministic algorithm which, on input a secret key k_{id}, a message $m \in \mathscr{M}$, and a tag t, outputs 1 (acceptance) or 0 (rejection).
- $(\tau_i)_{i\in[u]} \leftarrow \mathrm{DSeqAgg}(T)$: DSeqAgg is a deterministic algorithm which, on input a sequence of tags $T = ((id_{\ell_i}, t_i))_{i\in[N]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_N}) \in \mathscr{S}$, outputs a tuple of sequential aggregate tags $(\tau_i)_{i\in[u]} \in \mathscr{T}_{agg}$.
- $J \leftarrow \mathrm{DSAVrfy}(K, M, (\tau_i)_{i\in[u]})$: DSAVrfy is a deterministic algorithm which, on input a set of key/id pairs $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$, a sequence of message/id pairs $((m_i, id_{\ell_i}))_{i\in[N]}$ for any $(id_{\ell_1}, \ldots, id_{\ell_N}) \in \mathscr{S}$, and an aggregate tag $(\tau_i)_{i\in[u]}$, outputs a list J consisting of IDs whose messages are invalid or their orders are invalid.

In the above definition, we note that the number of aggregated MAC tags is fixed, namely always N, from the consistency with the definition of aggregate MACs with group testing as in [9].

Here, we explain what DSAVrfy outputs by using the following illustration: Let $\mathsf{ID} := \{id_1, id_2, \ldots, id_9\}$ (i.e., $N = 9$). Suppose that for a tuple of tags $(\tau_i)_{i\in[u]}$ on $M = ((id_1, m_1), (id_2, m_2), \ldots, (id_9, m_9))$, DSAVrfy outputs $J = \{id_1, id_9\}$. Then, it means that (id_1, m_1) and (id_9, m_9) are invalid (i.e., the messages themselves are invalid or its ordering is invalid), and that other pairs of ID/message and their order are valid (i.e., a correct sequence of messages is $(*, m_2, m_3, \ldots, m_8, *)$ but a verifier does not know $*$).

For a SAMD, we require that the following condition (i.e., correctness) holds:

- For all $id \in \mathsf{ID}$, all $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$ and all $m \in \mathscr{M}$, we have $1 \leftarrow \mathrm{Vrfy}(k_{id}, m, \mathrm{Tag}(k_{id}, m))$.
- For all $id \in \mathsf{ID}$, all $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$ and all $m \in \mathscr{M}$, for any $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ and any $M = ((m_i, id_{\ell_i}))_{i\in[N]}$ such that $(id_{\ell_1}, \ldots, id_{\ell_N}) \in \mathscr{S}$, we have $\emptyset \leftarrow \mathrm{DSAVrfy}(K, M, (\tau)_{i\in[u]})$, where $(\tau_i)_{i\in[u]} \leftarrow \mathrm{DSeqAgg}(((id_{\ell_i}, \mathrm{Tag}(k_{id_{\ell_i}}, m_i)))_{i\in[N]})$.

We define the security of SAMD called aggUF-CMA security and identifiability as follows.

Definition 5 (C-aggUF-CMA security)**.** *A SAMD scheme* $(\mathrm{KGen}, \mathrm{Tag}, \mathrm{Vrfy}, \mathrm{DSeqAgg}, \mathrm{DSAVrfy})$ *meets* C-aggUF-CMA *security if for any PPT adversary* $\mathscr{A}$ *against* SAMD*, the advantage* $Adv^{\text{agg-uf}}_{SAMD,\mathscr{A}}(\lambda) := \Pr[\mathscr{A}\ wins]$ *is negligible, where* $[\mathscr{A}\ wins]$ *is the event that* $\mathscr{A}$ *wins in the following game:*

Setup: *A challenger generates a set of secret-key/ID pairs* $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ *by using the KGen algorithm. Then, it sets lists* $\mathscr{L}_{Cor} = \emptyset$ *and* $\mathscr{L}_{SA} = \emptyset$*.*

Corrupt: *The corrupt oracle* $\mathsf{Corrupt}(\cdot)$ *takes an ID* $id \in \mathsf{ID}$ *as input and returns the secret key* k_{id} *and sets* $\mathscr{L}_{Cor} \leftarrow \mathscr{L}_{Cor} \cup \{id\}$*, where* $\mathscr{L}_{Cor}$ *means a list of IDs whose corresponding secret keys are known by an adversary. The number of queries submitted by* $\mathscr{A}$ *is at most* C*.*

Tagging: *The aggregate tagging oracle* $\mathsf{DSATag}_K(\cdot)$ *takes a sequence of message/ID pairs* $M = ((m_i, id_{\ell_i}))_{i\in[N]}$ *such that* $(id_{\ell_1}, \ldots, id_{\ell_N}) \in \mathscr{S}$ *and the number of not corrupted tags is at least* $N - C$*. Then, it does the following:*

1. *Compute* $t_i \leftarrow \mathrm{Tag}(k_{id_{\ell_i}}, m_i)$ *for all* $i \in [N]$*,*
2. *Output* $(\tau_i)_{i\in[u]} \leftarrow \mathrm{DSeqAgg}(((id_{\ell_i}, t_i))_{i\in[N]})$*,*
3. *Set* $\mathscr{L}_{SA} \leftarrow \mathscr{L}_{SA} \cup \{M\}$*.*

The number of queries which $\mathscr{A}$ *submits is at most* $Q = \mathrm{poly}(\lambda)$*, and* $\mathscr{A}$ *is not allowed to access* $\mathrm{Corrupt}(\cdot)$ *after accessing* $\mathrm{DSATag}_K(\cdot)$*.*

Output: *When* $\mathscr{A}$ *outputs* $M^* = ((m^*_i, id_{\ell^*_i}))_{i\in[N]}$ *and* $(\tau^*_i)_{i\in[u]}$*,* $\mathscr{A}$ *wins if the following holds:*

- $\emptyset \leftarrow \mathrm{DSAVrfy}(K, M^*, (\tau_i^*)_{i\in[u]})$,
- $(id_{\ell_1^*}, \ldots, id_{\ell_N^*}) \in \mathscr{S}$ *such that the number of not-corrupted IDs is at least* $N-C$,
- $M^* \notin \mathscr{L}_{SA}$ *such that* $id_{\ell_1^*}, id_{\ell_N^*} \notin \mathscr{L}_{Cor}$.

Next, we define identifiability of SAMD. Unlike the aggUF-CMA security game, adversaries are allowed to access the tagging oracle of each MAC in the game of the identifiability. This is because the purpose of adversaries trying to break identifiability is to make a verifier regard valid messages as invalid ones, or to make him/her regard invalid messages as valid ones. On the other hand, the purpose of adversaries trying to break aggUF-CMA security is to generate forgeries of sequential aggregate tags.

We formalize the identifiability of SAMD based on the definition of the identifiability in [9]. That is, the identifiability is defined by two kinds of notions, completeness and soundness. Intuitively, completeness prevents adversaries from making a verifier regard valid messages as invalid ones. On the other hand, soundness prevents adversaries from making a verifier regard invalid messages as valid ones.

Definition 6 (C-Identifiability)**.** *A SAMD scheme* $\mathrm{SAMD} = (\mathrm{KGen}, \mathrm{Tag}, \mathrm{Vrfy}, \mathrm{DSeqAgg}, \mathrm{DSAVrfy})$ *meets* C-identifiability *if it satisfies* C-completeness *and* C-soundness *which are defined below. Let* $\mathscr{A}$ *be a PPT adversary against* SAMD*, and we consider the following game:*

Setup: *A challenger generates a set of secret-key/ID pairs* $K = \{(k_{id_i}, id_i)\}_{i\in[N]}$ *by using the KGen algorithm. Then, it sets lists* $\mathscr{L}_{Cor} = \emptyset$ *and* $\mathscr{L}_{Tag} = \emptyset$.

Corrupt: *The corrupt oracle* $\mathsf{Corrupt}(\cdot)$ *takes an ID* $id \in \mathsf{ID}$ *as input and returns the secret key* k_{id} *and sets* $\mathscr{L}_{Cor} \leftarrow \mathscr{L}_{Cor} \cup \{id\}$*, where* $\mathscr{L}_{Cor}$ *means a list of IDs whose corresponding secret keys are known by an adversary. The number of queries submitted by* $\mathscr{A}$ *is at most* C.

Tagging: *The tagging oracle* $\mathsf{Tag}_K(\cdot)$ *takes a message/ID pair* (m_i, id_{ℓ_i})*, and returns* $t_i \leftarrow \mathrm{Tag}(k_{id_i}, m_i)$ *and sets* $\mathscr{L}_{Tag} \leftarrow \mathscr{L}_{Tag} \cup \{(m_i, id_{\ell_i})\}$*. The number of queries which* $\mathscr{A}$ *submits is at most* $Q = \mathrm{poly}(\lambda)$*.* $\mathscr{A}$ *is allowed to access* $\mathrm{Corrupt}(\cdot)$ *after accessing* $\mathrm{Tag}_K(\cdot)$ *in this security game.*

Output: *When* $\mathscr{A}$ *outputs* $((m_i^*, id_{\ell_i^*}, t_i^*)_{i\in[N]}$*, the challenger runs* $J \leftarrow \mathrm{DSAVrfy}(K, M^*, (\tau_i^*)_{i\in[u]})$*, where* $M^* \leftarrow ((m_i^*, id_{\ell_i^*}))_{i\in[N]}$ *and* $(\tau_i^*)_{i\in[u]} \leftarrow \mathrm{DSeqAgg}(((id_{\ell_i^*}, t_i^*))_{i\in[N]})$.

C-completeness *and* C-soundness *are defined as follows:*

- *Completeness:* SAMD *meets* C-completeness*, if for any PPT adversary* $\mathscr{A}$*, the advantage* $Adv^{\text{ident-c}}_{\mathrm{SAMD},\mathscr{A}}(\lambda)$ *of* $\mathscr{A}$ *is negligible, where* $Adv^{\text{ident-c}}_{\mathrm{SAMD},\mathscr{A}}(\lambda)$ *is a probability that* $J \cap \{id_{\ell_i^*} \mid 1 \leftarrow \mathsf{Vrfy}(k_{id_{\ell_i^*}}, m_i^*, t_i^*), 1 \le i \le N\} \neq \emptyset$ *holds.*
- *Soundness:* SAMD *meets* C-soundness*, if for any PPT adversary* $\mathscr{A}$*, the advantage* $Adv^{\text{ident-s}}_{\mathrm{SAMD},\mathscr{A}}(\lambda)$ *of* $\mathscr{A}$ *is negligible, where* $Adv^{\text{ident-s}}_{\mathrm{SAMD},\mathscr{A}}(\lambda)$ *is a probability that* $\{id_{\ell_i^*} \mid 0 \leftarrow \mathsf{Vrfy}(k_{id_{\ell_i^*}}, m_i^*, t_i^*), 1 \le i \le N\} \backslash J \neq \emptyset$ *holds.*

In addition, we explain that the above security definition is enough regarding identifiability because one may think that it is necessary to consider the case where outside adversaries replace orders of messages. Regarding the attacks which adversaries break identifiability by replacing orders of messages, we define the following two types of attacks:

(1). Attack that an adversary tries to make a verifier regard a sequence of messages with an incorrect order as the one with a correct order.

(2). Attack that an adversary tries to make a verifier regard a sequence of messages with a correct order as the one with an incorrect order.

Regarding the attack (1), the aggUF-CMA security of SAMD obviously guarantees that adversaries cannot attain their purpose. As for the attack (2), we cannot prevent this attack because adversaries can attain the purpose by tampering tags corresponding to targeted message/ID pairs. However, a verifier can exactly detect invalid message/ID pairs from the above identifiability.

5.2 Construction of SAMD

We construct a SAMD scheme HSAMD in the random oracle model from any MAC scheme and any NGT protocol with a group testing matrix. Specifically, the following primitives are used to construct a SAMD scheme:

- A MAC scheme $\mathrm{MAC} = (\mathrm{MAC.KGen}, \mathrm{MAC.Tag}, \mathrm{MAC.Vrfy})$.
- A group testing matrix $\boldsymbol{G} = (g_{i,j}) \in \{0,1\}^{u \times N}$. For every $i \in [u]$, the i-th row of $\boldsymbol{G}$ is denoted by $\boldsymbol{g}_i$. In addition, Let $I(\boldsymbol{G}, i)$ be the set of j ($1 \leq j \leq n$) such that $g_{i,j} = 1$ for every $i \in [u]$.
- A hash function $H : \{0,1\}^* \to \{0,1\}^n$.

Then, $\mathrm{HSAMD} = (\mathrm{KGen}, \mathrm{Tag}, \mathrm{Vrfy}, \mathrm{DSeqAgg}, \mathrm{DSAVrfy})$ is constructed as follows:

- $k_{id} \leftarrow \mathrm{KGen}(1^\lambda, id)$: Output a secret key $k_{id} \leftarrow \mathrm{MAC.KGen}(1^\lambda)$ for every ID id.
- $t \leftarrow \mathrm{Tag}(k_{id}, m)$: Output a tag $t \leftarrow \mathrm{MAC.Tag}(k_{id}, m)$.
- $1/0 \leftarrow \mathrm{Vrfy}(k_{id}, m, t)$: Output $b \leftarrow \mathrm{MAC.Vrfy}(k_{id}, m, t)$.
- $(\tau_i)_{i \in [u]} \leftarrow \mathrm{DSeqAgg}(T)$: For $T = ((id_{\ell_1}, t_1), \ldots, (id_{\ell_N}, t_N))$, do the following:
  1. For an ID-string $\mathbf{s} = (id_{\ell_1}, \ldots, id_{\ell_N}) \in \mathscr{S}$ and $I(\boldsymbol{G}, i) = \{j_1, j_2, \ldots, j_k\}$, we define a string $\mathbf{s}(\boldsymbol{G}, i) := (id_{\ell_{j_1}}, id_{\ell_{j_2}}, \ldots, id_{\ell_{j_k}})$ consisting of IDs keeping the order of $\mathbf{s}$. Then, for every $i \in [u]$, $\tau_i \leftarrow H(t_{j_1}, t_{j_2}, \ldots, t_{j_k})$.
  2. Output $(\tau_i)_{i \in [u]}$
- $J \leftarrow \mathrm{DSAVrfy}(K, M, (\tau_i)_{i \in [u]})$: Verify $M = ((m_i, id_{\ell_i}))_{i \in [N]}$ and $(\tau_i)_{i \in [u]}$ as follows:
  1. For $i = 1, 2, \ldots, u$, do the following: $J \leftarrow J \backslash \{id_{\ell_j}\}$ for all $1 \leq j \leq N$ with $g_{i,j} = 1$ if $\tau_i = H(t_{j_1}, t_{j_2}, \ldots, t_{j_k})$ holds, where $(t_{j_1}, \ldots, t_{j_k}) = (\mathrm{Tag}(k_{id_{\ell_{j_1}}}, m_1), \ldots, \mathrm{Tag}(k_{id_{\ell_{j_k}}}, m_k))$ and $(id_{\ell_{j_1}}, \ldots, id_{\ell_{j_k}}) = \mathbf{s}(G, i)$.
  2. Output J.

We show the security of HSAMD as follows.

Theorem 3. *If* MAC *meets* UF-CMA *security, the resulting* HSAMD *meets* $(N-u)$-aggUF-CMA *security in the random oracle model. In addition, if a group testing matrix* $\boldsymbol{G}$ *is* d-disjunct*, then* HSAMD *meets* $(N-u)$-identifiability *in the random oracle model.*

Proof. Let $\mathscr{A}$ be a PPT adversary against HSAMD. Let Q_h be the number of queries issued to the random oracle, and let Q_t be the number of queries issued to the tagging oracle.

First, regarding the aggUF-CMA security of HSAMD, let Forge be the event that $\mathscr{A}$ breaks HSAMD by making a forgery of the underlying MAC, and let Coll be the event that $\mathscr{A}$ finds a collision of the random oracle H. Then, we have $Adv^{agg-uf}_{\mathrm{HSAMD},\mathscr{A}}(\lambda) := \Pr[\mathsf{Forge}] \leq \Pr[\mathsf{Coll}] + \Pr[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}]$. $\Pr[\mathsf{Coll}] \leq \frac{(Q_h + uQ_t)^2}{2^{n+1}}$ holds since the number of queries submitted to the random oracle is at most $(Q_h + uQ_t)$. As for

the event $[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}]$, we can construct a PPT algorithm $\mathscr{F}$ breaking the UF-CMA security of MACs, in the same way as the proof of Theorem 2. Then, $\Pr[\mathsf{Forge} \wedge \overline{\mathsf{Coll}}] \leq N \cdot Adv^{\text{uf-cma}}_{MAC,\mathscr{F}}(\lambda)$. Therefore, we obtain

$$Adv^{\text{agg-uf}}_{HSAMD,\mathscr{A}}(\lambda) \leq N \cdot Adv^{\text{uf-cma}}_{MAC,\mathscr{F}}(\lambda) + \frac{(Q_h + uQ_t)^2}{2^{n+1}}.$$

We notice that each τ_i needs to include at least an ID such that $id \notin \mathscr{L}_{Cor}$. This is because if all inputs of H are corrupted IDs, there is a possibility that the aggUF-CMA security is broken in the same way as the proof of Proposition 1. Thus, the number of corrupted IDs is at most $(N-u)$ since the number of not corrupted IDs is at least u.

Next, we consider the identifiability of HSAMD. Regarding completeness, HSAMD meets $(N-u)$-completeness for at most d invalid messages from the property of a d-disjunct matrix. As for $(N-u)$-soundness, adversaries need to find a collision of the random oracle in order to break this security. The success probability is at most $\frac{Q_h^2}{2^{n+1}}$, since the number of calling the random oracle is at most Q_h. Hence, we have the following advantage:

$$Adv^{\text{ident-s}}_{HSAMD,\mathscr{A}}(\lambda) \leq \frac{Q_h^2}{2^{n+1}}.$$

From the above discussion, the proof is completed. □

6 Application: Sending Long Data by Data-Partitioning

Suppose that a device wants to send a long message in a wireless network, but the message is too long to directly transmit because of restrictions in the network. In this case, we usually utilize a data partitioning method to transmit the long data: We first divide a long message M into (at most) N pieces $m_1, m_2, \ldots, m_N$ (e.g., each piece may be called a packet); For each divided part m_j $(1 \leq j \leq N)$, the device generates a MAC tag $t_j \leftarrow \text{Tag}(k, (m_j, j))$; The device sends $((m_j, j), t_j)$ for $j = 1, 2, \ldots, N$ by possibly different paths in the network; A receiver obtains $\{(m_j, t_j)\}_{j \in [N]}$, where we assume that divided parts $m_1, m_2, \ldots, m_N$ do not necessarily reach with the correct order (e.g., some of which may be delayed in the network) and he will check the validity of both divided data and their ordering to correctly recover the message M. In this situation, we note that N tags are transmitted in the wireless network, which may cause a traffic problem if there are an enormous number of devices connected to the network and each device wants to send a long message. Our idea is to apply SAMD in the previous section in order to reduce the numbers of tags for divided data, so that we resolve the problem by reducing the amount of tags with the aggregation technique without changing the structure of the underlying MACs.

Formally, suppose that an existing authentication protocol utilizes a MAC scheme MAC=(KGen, Tag, Vrfy) as follows, where a secret key $k \leftarrow \text{KGen}(1^\lambda)$ is already generated and installed in a device and such secret keys are generally different in devices:

Conventional Protocol.

- Transmission by Data-Partitioning:
  1. For a long message M, generate divided messages $(m_1, 1), (m_2, 2), \ldots, (m_N, N)$ from M by a data partitioning technique.
  2. For each (m_j, j), generate its tag $t_j \leftarrow \text{Tag}(k, (m_j, j))$. [1]

[1] We do not consider the case $t_j \leftarrow \text{Tag}(k, m_j)$, because an adversary can replace j with $j' \neq j$ and hence the protocol is insecure in this case.

3. It transmits $((m_j, j), t_j)$ for $j = 1, 2, \ldots, N$ by possibly different paths in the network.

- Verification and Message Recovery: On receiving $\{((m_j, j), t_j)\}_{j \in [N]}$, it checks validity of both divided data and their ordering, and we can recover the message as follows:
  - If $1 \leftarrow \mathrm{Vrfy}(K, (m_j, j), t_j)$ for every $j \in [N]$, M is recovered by the sequential data $M = (m_1, m_2, \ldots, m_N)$.
  - Suppose that $0 \leftarrow \mathrm{Vrfy}(K, (m_j, j), t_j)$ for $j \in J = \{j_1, j_2, \ldots, j_s\} \subseteq [N]$ and $1 \leftarrow \mathrm{Vrfy}(K, (m_j, j), t_j)$ for every $j \in [N] \setminus J$. Then, it requests the device to transmit the packets $((m_j, j), t_j)$ for all $j \in J$ again, and it checks validity of (m_j, j) with Vrfy. This procedure is repeated until all packets (m_j, j) $(j \in J)$ are verified as valid. If all packets are verified as valid, M is recovered by the sequential data $M = (m_1, m_2, \ldots, m_N)$.

In order to resolve a traffic problem, we propose to use SAMD=(KGen, Tag, Vrfy, DSeqAgg, DSAVrfy) as an application in this scenario, where KGen, Tag, and Vrfy are the algorithms of the underlying MAC. We consider to embed DSeqAgg algorithm into a device and DSAVrfy algorithm into an verification protocol/system as application. Then, we propose the following protocol:

Our Protocol.

- Transmission by Data-Partitioning:

  1 and 2. The same in the conventional protocol.

  3. Compute $(\tau_i)_{i \in [u]} \leftarrow \mathrm{DSeqAgg}((t_i, i))_{i \in [N]})$, and then transmit N pieces $((m_1, 1), \tau_1)$, $((m_2, 2), \tau_2)$, $\ldots$, $((m_u, u), \tau_u)$, $(m_{u+1}, u+1), \ldots, (m_N, N))$ by possibly different paths in the network, where we note that a tag is attached only to $(m_1, 1), \ldots, (m_u, u)$, and we assume $u < N$.

- Verification and Message Recovery: On receiving $((m_1, 1), \tau_1), ((m_2, 2), \tau_2), \ldots, ((m_u, u), \tau_u), (m_{u+1}, u+1), \ldots, (m_N, N))$, it checks validity of both divided data and their ordering, and we can recover the message as follows:
  - If $\emptyset \leftarrow \mathrm{DSAVrfy}(k, \tilde{M}, (\tau_i)_{i \in [u]})$ where $\tilde{M} = ((m_1, 1), (m_2, 2), \ldots, (m_N, N))$, M is recovered by the sequential data $M = (m_1, m_2, \ldots, m_N)$.
  - Suppose that $J \leftarrow \mathrm{DSAVrfy}(k, \tilde{M}, (\tau_i)_{i \in [u]})$ for some $J = \{j_1, j_2, \ldots, j_s\} \neq \emptyset$. Then, it requests the device to transmit the packets $((m_j, j), t_j)$ for all $j \in J$ again, and it checks validity of (m_j, j) with Vrfy of the underlying MAC[2]. This procedure is repeated until all packets (m_j, j) $(j \in J)$ are verified as valid. If all packets are verified as valid, M is recovered by the sequential data $M = (m_1, m_2, \ldots, m_N)$.

It should be noted that the number of MAC-tags transmitted by data-partitioning in the initial phase is u in our protocol, while the number of MAC-tags transmitted is N in the conventional protocol. Hence, we can resolve the traffic problem effectively if $u \ll N$. In addition, we note that in each device, the same key k is used for generating N MAC-tags $t_1, t_2, \ldots, t_N$. Therefore, we need to apply a SAMD that meets 0-aggUF-CMA and 0-identifiability.

7 Conclusion

In this paper, we introduced a new model of sequential aggregate MACs (SAMACs) where an aggregation algorithm generates a sequential aggregate tag depending only on multiple and independent MAC

[2]This is because it seems to be plausible to assume $|J| \ll N$.

tags without any secret-key, and we formally defined security in this model. Our model and security are quite different from those of previous works [6, 8, 13]. In addition, we proposed two generic constructions, SAMAC1 and SAMAC2, starting from any MACs, with formal security proofs. And, we compared the existing ones and ours in terms of universal applicability, security, and efficiency. As a result, SAMAC2 is superior from all aspects of evaluation items, though the security proof is given in the random oracle model.

In the above sequential aggregate MACs, if a sequence of messages are rejected in our sequential aggregate MACs, we cannot identify which message has been invalid (e.g., some of them was forged, or their order was wrong). Therefore, by extending SAMACs, we introduced a sequential aggregate MAC which has detecting functionality (SAMD). The SAMD enables us to specify an invalid message or an invalid order of a certain message. We also formalized the security of SAMD. Moreover, we provided a generic construction of SAMD in the random oracle model from any MAC scheme and any non-adaptive group testing (NGT) protocol, and we prove that the resulting SAMD meets our security definition if the underlying MAC meets UF-CMA security and the NGT is given by a d-disjunct matrix.

Acknowledgements

The authors would like to thank the anonymous referees of ProvSec 2018 for their helpful comments. This research was conducted under a contract of Research and Development for Expansion of Radio Wave Resources funded by the Ministry of Internal Affairs and Communications, Japan.

References

[1] M. Bellare. New proofs for NMAC and HMAC: security without collision resistance. *Journal of Cryptology*, 28(4):844–878, October 2015.

[2] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Proc. of the 16th Annual International Cryptology Conference (CRYPTO'96), Santa Barbaera, California, USA*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, August 1996.

[3] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, December 2000.

[4] R. Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943.

[5] D.-Z. Du and F. K. Hwang. *Combinatorial Group Testing and Its Applications (2nd Edition)*, volume 12 of *Series on Applied Mathematics*. World Scientific, 2000.

[6] O. Eikemeier, M. Fischlin, J. Götzmann, A. Lehmann, D. Schröder, P. Schröder, and D. Wagner. History-free aggregate message authentication codes. In *Proc. of the 7th International Conference on Security and Cryptography for Networks (SCN'10), Amalfi, Italy*, volume 6280 of *Lecture Notes in Computer Science*, pages 309–328. Springer-Verlag, September 2010.

[7] D. Eppstein, M. T. Goodrich, and D. S. Hirschberg. Improved combinatorial group testing algorithms for real-world problem sizes. *SIAM Journal on Computing*, 36(5):1360–1375, January 2007.

[8] S. Hirose and H. Kuwakado. Forward-secure sequential aggregate message authentication revisited. In *Proc. of the 8th International Conference on Provable Security (ProvSec'14), Hong Kong, China*, volume 8782 of *Lecture Notes in Computer Science*, pages 87–102. Springer-Verlag, October 2014.

[9] S. Hirose and J. Shikata. Non-adaptive group-testing aggregate MAC scheme. In *Proc. of the 14th International Conference on Information Security Practice and Experience (ISPEC'18), Tokyo, Japan*, volume 11125 of *Lecture Notes in Computer Science*, pages 357–372. Springer-Verlag, September 2018.

[10] F. K. Hwang. A method for detecting all defective members in a population by group testing. *Journal of the American Statistical Association*, 67(339):605–608, September 1972.

[11] J. Katz and A. Y. Lindell. Aggregate message authentication codes. In *Proc. of the Cryptographers' Track at the RSA Conference (CT-RSA'08), San Francisco, California, USA*, volume 4964 of *Lecture Notes in Computer Science*, pages 155–169. Springer-Verlag, April 2008.

[12] C. H. Li. A sequential method for screening experimental variables. *Journal of the American Statistical Association*, 57(298):455–477, June 1962.

[13] D. Ma and G. Tsudik. Extended abstract: Forward-secure sequential aggregate authentication. In *Proc. of the 2007 IEEE Symposium on Security and Privacy (SP'07), Berkeley, California, USA*, pages 86–91. IEEE, May 2007.

[14] N. I. of Standards and Technology. Recommendation for block cipher modes of operation: the cmac mode for authentication. Technical Report NIST Special Publication 800-38G, National Institute of Standards and Technology, 2005.

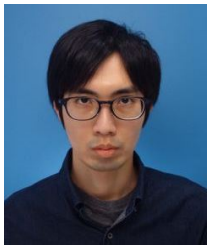
[15] E. Porat and A. Rothschild. Explicit non-adaptive combinatorial group testing schemes. In *Proc. of the 35th International Colloquium on Automata, Languages, and Programming (ICALP'08), Reykjavik, Iceland*, volume 5125 of *Lecture Notes in Computer Science*, pages 748–759. Springer-Verlag, July 2008.

[16] S. Sato, S. Hirose, and J. Shikata. Generic construction of sequential aggregate macs from any macs. In *Proc. of the 12th International Conference on Provable Security (ProvSec'18), Jeju, South Korea*, volume 11192 of *Lecture Notes in Computer Science*, pages 295–312. Springer-Verlag, October 2018.

[17] N. Thierry-Mieg. A new pooling strategy for high-throughput screening: the shifted transversal design. *BMC Bioinformatics*, 7(28):1–13, January 2006.

[18] S. Tomita, Y. Watanabe, and J. Shikata. Sequential aggregate authentication codes with information theoretic security. In *Proc. of the 2016 Annual Conference on Information Science and Systems (CISS'16), Princeton, New Jersey, USA*, pages 192–197. IEEE, March 2016.

Author Biography



Shingo Sato received the MS degree in Information Science from Yokohama National Univdersity, Japan in 2017. He is currently a PhD student in Yokohama National University. His research interests include cryptography and information security.

Shoichi Hirose Shoichi Hirose received the B.E., M.E. and D.E. degrees in information science from Kyoto University, Kyoto, Japan, in 1988, 1990 and 1995, respectively. From 1990 to 1998, he was a research associate at Faculty of Engineering, Kyoto University. From 1998 to 2005, he was a lecturer at Graduate School of Informatics, Kyoto University. From 2005 to 2009, he was an associate professor at Faculty of Engineering, University of Fukui. From 2009 to 2016, he was a professor at Graduate School of Engineering, University of Fukui. From 2016, he is a professor at Faculty of Engineering, University of Fukui. His current interests include cryptography and information security.

Junji Shikata received the B.S. and M.S. degrees in mathematics from Kyoto University, Kyoto, Japan, in 1994 and 1997, respectively, and the Ph.D. degree in mathematics from Osaka University, Osaka, Japan, in 2000. From 2000 to 2002 he was a Postdoctoral Fellow at the Institute of Industrial Science, the University of Tokyo, Tokyo, Japan. Since 2002 he has been with the Graduate School of Environment and Information Sciences, Yokohama National University, Yokohama, Japan. From 2008 to 2009, he was a visiting researcher at the Department of Computer Science, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland. Currently, he is a Professor of Yokohama National University. His research interests include cryptology, information theory, theoretical computer science, and computational number theory. He received several awards including the Wilkes Award 2006 from the British Computer Society, and the Young Scientists' Prize, the Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology in Japan in 2010.